REMARKS

Claims 1-34 stand rejected. Claims 1, 3-6, 9-10, 21, and 32 have been amended. Claims 1-34 are presently pending.

The Applicants thank the Examiner for the examination of the present application.

Figures 1 and 2 have been amended as suggested by the Examiner. Substitute drawing sheets have been attached.

No new matter has been added by the present amendment. The new and amended claims are new, non-obvious, and useful, and are supported throughout the originally filed disclosure. Reconsideration of the present application is requested in light of the amendment, and the remarks given below.

REJECTION OF CLAIMS 1-34 UNDER 35 U.S.C. § 102(A) OVER BACKGROUND

The Examiner rejected claims 1-34 under 35 U.S.C. § 102 (a) over the Background section of the present application. Claims 1, 3-5, 9-10, 21 and 32 have been amended to clarify the present invention recited therein. Claim 6 was amended to correct a typographical error. For at least the following reasons the Background section of the present application does not anticipate claims 1-34.

Claim 1, as amended, recites in part:

> while the task still holds the at least one resource, lowering a current priority of the task when testing the priority inheritance variable indicates that the task holds no resources that are involved in a priority inheritance.

The ratchet algorithm, described in the Background section, only lowers the priority of a task whose priority has been raised by priority inheritance after all mutual exclusion resources held by the task have been released. The count used in the ratchet algorithm is a count of all mutual exclusion resources held by the task which is inheriting a higher priority, not a count of the number of resources held for which higher base priority tasks are waiting. This means that a task whose priority has been elevated by priority inheritance, in some situations, will continue to have an elevated inherited priority even after the release all resources for which higher priority tasks

are waiting. The priority of a task which has inherited a higher priority will not be lowered while the task still retains any mutual exclusion resources. In contrast, amended claim 1 recites that the priority of the task is lowered "while the task still holds the first resource", meaning the lowering to a base priority occurs while the task still retains mutual exclusion resources. This will never occur in the traditional ratchet algorithm. For at least this reason, the ratchet algorithm recited in the Background section of the present application does not anticipate amended claim 1.

Claim 2 depends from claim 1, and thus should be patentable for at least the reasons given above for Claim 1. Moreover, claim 2 recites

> the priority inheritance variable is configured to have a value indicative of the number of resources held by the task that higher priority tasks are waiting to receive.

The conventional ratchet algorithm uses a count of the mutual exclusion resources held by the task which inherits a higher priority. In contrast, claim 2 recites the use of a priority inheritance variable configured to track the number of resources held by the task that higher priority tasks are waiting to receive. This avoids the situation described above, where the low priority task retains its inherited high priority even after all mutual exclusion resources needed by higher priority tasks have been released. This may result in greatly improved performance in a system with mutual exclusion resources.

Claims 3-6, and 9-11 depend, directly or indirectly, from claim 1, and thus should be patentable for at least the reasons given above for claim 1.

Claim 7 depends from claim 1, and thus should be patentable for at least the reasons given above for claim 1. The Examiner argues that 4:26-28 of the specification teaches this feature. The Applicants respectfully traverse. As an initial matter, the Applicants respectfully point out that "the task" recited in claim 7 is the task which inherits a higher priority from "the higher priority task". Claim 7 recites that the priority inheritance variable is adjusted when the higher priority task, the task from which priority is inherited, is deleted. In the ratchet algorithm, the count is adjusted when the task that inherited the higher priority releases resources. The Background section does not discuss adjusting priority for the deletion of the waiting higher priority task. In fact, this is another example where the claimed invention is superior to the ratchet algorithm. In the ratchet algorithm, the task which inherited high priority would continue to run at high priority until it gives up its mutual exclusion resources, regardless of whether the waiting high priority task from which priority was inherited was no longer in need of the

resources, e.g., when the high priority task is deleted as recited in claim 7.

Claim 8 is patentable for reasons similar to claim 7. Moreover, To anticipate a claim, the reference must teach every element of the claim. *See* MPEP 2131. The identical invention must be shown in as complete detail as is contained in the claim. *See id.* (Citing *Richardson v. Suzuki* Motor Co., 868 F.2d 1226, 1236 (Fed. Cir. 1989)). The Examiner points to the "list" or "stack" algorithm discussed in the background section. This algorithm is distinct from the ratchet algorithm also discussed in the background section. There is no "incrementing" or "counting" in the list algorithm. Rather a list of resource requests, potentially unbounded, is tracked. There is no teaching or suggestion that the list algorithm is used in combination with the count of the ratchet algorithm. Thus the Section 102 rejection is inappropriate, as neither approach described in the Background section includes all the limitations of claim 8.

Claim 12 stands rejected over the Background section of the present application. The Applicants respectfully traverse. In particular, claim 12 recites "the priority inheritance variable associated with the task and configured to be indicative of the number of resources held by the task that higher priority tasks are waiting to receive." The section cited by the Examiner, 4:25-33 discusses the ratchet algorithm, where there is a count is made of mutual exclusion resources held by the low priority task. This is not a count of the resources held by the low priority task that higher priority tasks are waiting to receive, but merely a count of all resources held by low priority task.

Claim 13 stands rejected over the Background section of the present application. Claim 13 recites a "priority inheritance variable configured to have a value indicate of the number of inversion safe mutual exclusion semaphores held by the task that higher priority tasks are waiting to receive." The ratchet algorithm described in the background section, maintains a count of ALL mutual exclusion resources held by a task which has inherited a higher priority, not a count of the resources which higher priority tasks are waiting to received. Accordingly, it does not anticipate Applicants' claim 13. Claims 14-20 depend from claim 13 and thus should be patentable for at least the reasons given above for claim 13.

Amended claim 21 recites:

> while the task still holds a second inversion safe mutual exclusion semaphore, setting the current priority of the task to a base priority value when the counter is decremented to a value that indicates that the task holds no inversion safe mutual

exclusion semaphores involved in priority inheritance.

The conventional ratchet algorithm cited by the Examiner only resets the priority of a task with an inherited priority when it has released all the mutual exclusion resources it holds. Thus, the task would never have its priority lowered when it still holds mutual exclusion resources, even though all the mutual exclusion resources needed by waiting higher priority tasks had been released.

Claim 22 recites "a mutual exclusion control mechanism configured to set a current priority of the task to a base priority value when the priority inheritance variable indicates that no higher priority tasks are blocked on resources held by the task." The ratchet algorithm, cited by the Examiner, resets the priority of a task which has inherited higher priority only when the tasks has given up all its mutual exclusion resources. In contrast, claim 22 recites that the priority is set to the base priority when higher priority tasks are no longer blocked on resources held by the task. Claim 23 depends from claim 22 and thus should be patentable for at least the reasons given above for claim 22.

Claim 24 recites a "mutual exclusion control mechanism configured to set a current priority of the task to a base priority value when the priority inheritance variable indicates that no higher priority tasks are blocked on inversion safe mutual exclusion semaphores held by the task." Claim 25 depends from claim 24 and thus should be patentable over the cited art for at least the reasons given above for claim 24.

Claim 26 recites a variable associated with the semaphore and configured to indicate whether a pending request for the semaphore has resulted in a priority inheritance. The Examiner cites 4:17 in the Background section of the present application. The Applicant has not located the recited structure in the cited portion of the specification, and respectfully submits that this feature is not taught by the ratchet algorithm described in the Background section of the present application and cited by the Examiner. The Applicants respectfully requests either a more detailed citation or further clarification, so that this rejection can be responded to more fully. Claim 27 depends from claim 26 and thus should be patentable for similar reasons.

Claim 28 recites "a mutual exclusion control mechanism configured to adjust the priority inheritance variable when the task releases the semaphore only if the variable indicates that a pending request for the semaphore has resulted in a priority inheritance." The ratchet algorithm

cited by the Examiner adjusts the counter variable whenever a mutual exclusion resource is released by a task that has inherited a higher priority. In contrast, claim 28 recites that, in response to a semaphore release, the priority inheritance variable is adjusted only if a variable associated with the released semaphore indicates the semaphore was involved in a pending request for in a priority inheritance. This feature is not taught or suggested by the ratchet algorithm. In fact, the ratchet algorithm keeps track of all released resources regardless of whether they caused a priority inheritance. Accordingly, claim 28 should be allowable over the ratchet algorithm described in the Background section of the present application. Claim 29 depends from claim 28 and should be allowable for at least similar reasons.

Claim 30 recites "setting the current priority of the task to a base priority value whenever no higher priority tasks are waiting to receive any of the resources held by the task and the task still holds at least one resource." The Examiner again cites the ratchet algorithm described on page 4 of the Applicants' specification. As discussed above, the ratchet algorithm will only reset a tasks priority when it releases all its resources. Accordingly, it will not reset the priority "whenever no higher priority tasks are waiting" as recited in Applicants' claim 30. Claim 31 has a similar recitation, but is restricted to inversion safe mutual exclusion semaphores.

Claim 32 is analogous to claim 1, and has been similarly amended. It should be allowable for reasons similar to claim 1. Claim 34 is analogous to claim 30, and should be allowable for similar reasons which should make it allowable over the cited art for reasons similar to those given above. Claim 33 contains the limitation "setting the current priority of the task to a base priority value when the counter is decremented to a value that indicates that the task holds no inversion safe mutual exclusion semaphores involved in priority inheritance". As discussed previously, this limitation is not present in the cited art.

For at least the above reasons, the Applicants respectfully submit that the rejection of claims 1-34 under 35 U.S.C. § 102 over the Background section should be withdrawn.

REJECTION OF CLAIMS 1,22,24,AND 26 UNDER 35 U.S.C. § 102 OVER MCDONALD '627

The Examiner rejected claims 1, 22, 24, and 26 under 35 U.S.C. 102(e) over U.S. Patent 6,56,627 to McDonald (hereinafter "McDonald '627"). For at least the following reasons,

McDonald '627 does not anticipate any of claims 1, 22, 24, and 26.

Claim 1 recites "while the task still holds the at least one resource, lowering a current priority of the task when testing the priority inheritance variable indicates that the task holds no resources that are involved in a priority inheritance." The flowcharts presented by McDonald '627 deal only with a single resource. The variable used by McDonald '627 indicates only when a particular resource has been freed, not when the task that has inherited priority holds "no resources that are involved in a priority inheritance." McDonald's '627 approach will not result in reducing an inherited priority of a task while the task still holds any resources. Accordingly, McDonald '627 does not teach or suggest the recited limitation of claim 1. Therefore, claim 1 should be allowable over McDonald '627.

Claims 3 and 4 depend from claim 1, and thus should be patentable over McDonald for at least the reasons given above for claim 1.

Claims 22 recites "a priority inheritance variable configured to indicate the number of resources that are held by the task and that at least one higher priority task is blocked on". The Examiner does not give a specific citation for this feature in McDonald. Applicant respectfully submits that this feature is neither taught nor suggested by McDonald.

Similarly claim 24 recites a priority inheritance variable associated with the task, the variable configured to indicate the number of inversion safe mutual exclusion semaphores that are held by the task and that at least one higher priority task is blocked on. The Examiner has not identified this feature in McDonald, and the Applicants respectfully submit that McDonald neither teacher or suggests this feature.

Claim 26 recites a variable, the variable associated with the semaphore and configured to indicate whether a pending request for the semaphore has resulted in a priority inheritance. The Examiner has not identified this feature in McDonald. Applicant respectfully submits this claim limitation is neither taught nor suggested by McDonald.

For at least the above reasons, the Applicants respectfully submit that the rejection of claims 1, 22, 24, and 26 under 35 U.S.C. § 102 over McDonald '627 should be withdrawn.

REJECTION OF CLAIMS 26-28 UNDER 35 U.S.C. § 102 OVER MURATA '628

The Examiner rejected claims 26-28 under 35 U.S.C. § 102(e) over U.S. Patent 6,560,628 to Murata (hereinafter "Murata '628"). For at least the following reasons, Applicant respectfully submits that Murata '628 does not anticipate any of claims 26-28.

Murata generally describes the list or stack algorithm which has also been described in the Background section of the present application and which has been discussed above. The disadvantage of this approach is that it requires unbounded resources, and the kernel run time increases linearly with the number of tasks in the system. The "inherit" variable of Murata, cited by the Examiner, is part of Murata's "thread data". It is not "associated with the semaphore". *See* Murata 8:50-58. Moreover, Murata's inherit variable does not "indicate whether a pending request for the semaphore has resulted in a priority inheritance." Rather, Murata's inherit variable indicates if a particular thread has received an inherited priority. This inherited priority could be from any resource, or in fact from multiple resources. Thus, the variable is not associated with a particular resource, and certainly not with a semaphore, as recited in Applicants' claim 26. Claims 27 and 28 depends from claim 26, and thus should be patentable over Murata for at least the reasons given above for claim 26.

With respect to 28, Applicant respectfully notes it includes two variables - the "variable" associated with the semaphore from parent claim 26 and the "priority inheritance variable" associated with the task literally in claim 28. Claim 28 recites

> a mutual exclusion control mechanism configured to adjust the priority inheritance variable when the task releases the semaphore only if the variable indicates that a pending request for the semaphore has resulted in a priority inheritance.

Here "the variable" is the variable from claim 26 associated with the semaphore. The Examiner has not identified where this feature is found in Murata, in particular the limitation that the priority inheritance variable associated with a semaphore is adjusted on a semaphore release only when the semaphore has resulted in a priority inheritance. Applicant respectfully submits that this feature is neither taught nor suggested by Murata.

For at least the above reasons, the Applicants respectfully submit that the rejection of claims 26-28 under 35 U.S.C. § 102 over Murata '628 should be withdrawn.

CONCLUSION

Each of the issues raised by the Examiner has been addressed. It is respectfully submitted that the present application is in condition for allowance. Prompt passage to issuance is requested.

Respectfully Submitted,

KENYON & KENYON

Dated: July 29, 2007

By: _____

Andrew L. Reibman
(Reg. No. 47,893)

One Broadway
New York, NY 10004
(212) 425-7200

CUSTOMER NO. 26646